3SH3 LAB4 L06 G08

Tingyu Shi(400253854)
Jiacheng Wu(400207981)

September 19, 2022

1 Inputs and Outputs

1.1 First set of input and output

This set of input and output are provided with the lab instruction.

1.1.1 Input

```
8 3
7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1 -1
```

1.1.2 Output

```
Number of pages: 8
Number of frames: 3
Size of the reference string: 20
Reference String:
7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1
```

FIFO

7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
7	0	1	2	2	3	0	4	2	3	0	0	0	1	2	2	2	7	0	1
-1	7	0	1	1	2	3	0	4	2	3	3	3	0	1	1	1	2	7	0
-1	-1	7	0	0	1	2	3	0	4	2	2	2	3	0	0	0	1	2	7
P	P	P	P		P	P	P	P	P	P			P	P			P	P	P

15 page-faults

Optimal

7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
7	7	7	2	2	2	2	2	2	2	2	2	2	2	2	2	2	7	7	7
-1	0	0	0	0	0	0	4	4	4	0	0	0	0	0	0	0	0	0	0
-1	-1	1	1	1	3	3	3	3	3	3	3	3	1	1	1	1	1	1	1
P	P	P	P		P		P			P			P				P		

9 page-faults

LRU

7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
7	7	7	2	2	2	2	4	4	4	0	0	0	1	1	1	1	1	1	1
-1	0	0	0	0	0	0	0	0	3	3	3	3	3	3	0	0	0	0	0
-1	-1	1	1	1	3	3	3	2	2	2	2	2	2	2	2	2	7	7	7
P	P	P	P		P		P	P	P	P			P		P		P		

12 page-faults

1.2 Second set of input and output

1.2.1 Input

```
8 3
4 7 6 1 7 6 1 2 7 2 -1
```

1.2.2 Output

```
Number of pages: 8
Number of frames: 3
Size of the reference string: 10
Reference String:
4 7 6 1 7 6 1 2 7 2
```

FIFO

4	7	6	1	7	6	1	2	7	2
4	7	6	1	1	1	1	2	7	7
-1	4	7	6	6	6	6	1	2	2
-1	-1	4	7	7	7	7	6	1	1
P	P	P	P				P	P	

6 page-faults

Optimal

4	7	6	1	7	6	1	2	7	2
4	4	4	1	1	1	1	2	2	2
-1	7	7	7	7	7	7	7	7	7
-1	-1	6	6	6	6	6	6	6	6
P	P	P	P				P		

5 page-faults

LRU

4	7	6	1	7	6	1	2	7	2
4	4	4	1	1	1	1	1	1	1
-1	7	7	7	7	7	7	2	2	2
-1	-1	6	6	6	6	6	6	7	7
P	P	P	P				P	P	

6 page-faults

1.3 Third set of input and output

1.3.1 Input

```
7 3
1 3 0 3 5 6 3 -1
```

1.3.2 Output

```
Number of pages: 7
Number of frames: 3
Size of the reference string: 7
Reference String:
1 3 0 3 5 6 3

FIFO
 1   3   0   3   5   6   3
-------------------------
 1   3   0   0   5   6   3
-1   1   3   3   0   5   6
-1  -1   1   1   3   0   5
 P   P   P       P   P   P
-------------------------
6 page-faults

Optimal
 1   3   0   3   5   6   3
-------------------------
 1   1   1   1   5   6   6
-1   3   3   3   3   3   3
-1  -1   0   0   0   0   0
 P   P   P       P   P
-------------------------
5 page-faults

LRU
 1   3   0   3   5   6   3
-------------------------
 1   1   1   1   5   5   5
-1   3   3   3   3   3   3
-1  -1   0   0   0   6   6
 P   P   P       P   P
-------------------------
5 page-faults
```

2 Code

NOTE: If you want to test the code, please change the file name in read_file function for different input files.
The following is the code.

```
#include <stdio.h>
#include <stdlib.h>

int N;  //number of pages
int M;  //number of frames
int RS[500];  //reference string array
int RS_len; //number of effective numbers in RS
int digit_max; //used to format printing


/* give a number, calculate the number of digit */
int digit_number(int number)
{
    if(number == 0)
    {
        return 1;
    }

    if(number == -1)
    {
        return 2;
    }

    int count=0;
    while(number != 0)
    {
       number = number / 10;
       count++;
    }
    return count;
}

/* calculate the maximum digit number among all the numbers to be
   displayed in the screen */
void get_max_digit()
{
    digit_max = digit_number(RS[0]);
    int i;
    for(i = 0; i < RS_len; i++)
    {
        if(digit_number(RS[i]) > digit_max)
        {
            digit_max = digit_number(RS[i]);
        }
    }

    /* -1 is included in the table, so if all numbers are single digit, digit_max = 2 */
    if (digit_max < 2)
    {
        digit_max = 2;
    }
}

/* given the starting address of an array and the index of the first digit of the number
   return the number */
int read_number(char* pos, int index)
{
    int i;
    int len = 0;
    int num = 0;
    int multipler = 1;

    char* temp_pos = pos;
    temp_pos += index;
```

```
    while(*temp_pos >= 48 && *temp_pos <= 57)
    {
        len++;
        temp_pos++;
    }
    temp_pos--;

    for(i = 0; i < len; i++)
    {
        num += (*temp_pos - 48) * multipler;
        multipler *= 10;
        temp_pos--;
    }

    return num;
}

void read_file()
{
    int i;
    char line[500];

    FILE * fpointer = fopen("sample.dat", "r");

    /* prepare N and M */
    fgets(line, 500, fpointer);
    N = read_number(&line[0], 0);
    i = 1;
    while (1)
    {
        if (line[i] >= 48 && line[i] <= 57 && line[i - 1] == ' ')
        {
            M = read_number(&line[0], i);
            break;
        }
        i++;
    }


    /* prepare reference string */
    for (i = 0; i < 500; i++)
    {
        RS[i] = -1;
    }
    int RS_index = 1;
    i = 1;
    fgets(line, 500, fpointer);
    RS[0] = read_number(&line[0], 0);
    while (1)
    {
        if (line[i] >= 48 && line[i] <= 57 && line[i - 1] == ' ')
        {
            RS[RS_index] = read_number(&line[0], i);
            RS_index++;
        }
        if(line[i] == '-')
        {
            break;
        }
        i++;
    }

    /* prepare RS_len */
    int count = 0;
    for(i = 0; i < 500; i++)
    {
        if (RS[i] != -1)
        {
            count ++;
        }
    }
```

```
    RS_len = count;
    fclose(fpointer);
}

void display_general_info()
{
    printf("Number of pages: %d\n", N);
    printf("Number of frames: %d\n", M);
    printf("Size of the reference string: %d\n", RS_len);
    printf("Reference String:\n");
    int i;
    for (i = 0; i < RS_len; i++)
    {
        printf("%d ", RS[i]);
    }
    printf("\n");
    printf("\n");
}

void display_table(int* table, int* pf_ind)
{
    int i;
    int j;
    int k;
    int temp_number;
    /* print reference string */
    for (i = 0; i < RS_len; i++)
    {
        for(j = 0; j < digit_max - digit_number(RS[i]); j++)
        {
            printf(" ");
        }
        printf("%d  ", RS[i]);
    }
    printf("\n");

    for(i = 0; i < (digit_max + 2) * RS_len; i++)
    {
        printf("-");
    }
    printf("\n");


    /* print table */
    for(i = 0; i < M; i++)
    {
        for (j = 0; j < RS_len; j++)
        {
            temp_number = *(table + (i * RS_len) + j);
            for(k = 0; k < digit_max - digit_number(temp_number); k++)
            {
                printf(" ");
            }
            printf("%d  ", temp_number);
        }
        printf("\n");
    }

    /* print hit or miss */
    for(i = 0; i < RS_len; i++)
    {
        if(*(pf_ind + i) == 1)
        {
            for(j = 0; j < digit_max - 1; j++)
            {
                printf(" ");
            }
            printf("P  ");
        }
        else
        {
            for(j = 0; j < digit_max + 2; j++)
```

```
            {
                printf("␣");
            }
        }
    }
    printf("\n");

    for(i = 0; i < (digit_max + 2) * RS_len; i++)
    {
        printf("-");
    }
    printf("\n");

    int counter = 0;
    for(i = 0; i < RS_len; i++)
    {
        if(*(pf_ind + i) == 1)
        {
            counter++;
        }
    }
    printf("%d␣page-faults\n", counter);
}

/* check if value exists in a col in the table  1--> exist  0 --> not exist */
int exist(int* table, int col, int value)
{
    int i;
    for(i = 0; i < M; i++)
    {
        if(*(table + i * RS_len + col) == value)
        {
            return 1;
        }
    }
    return 0;
}

/* push at the back and pop up the first
   used in FIFO*/
void push(int* table, int col, int value)
{
    int i;
    for(i = M - 2; i >= 0; i--)
    {
        *(table + (i + 1) * RS_len + col) = *(table + i * RS_len + col);
    }
    *(table + 0 * RS_len + col) = value;
}

/* copy one col to the next col */
void copy_col(int* table, int col)
{
    if(col == RS_len - 1)
    {
        return;
    }

    int i;
    for (i = 0; i < M; i++)
    {
        *(table + i * RS_len + col + 1) = *(table + i * RS_len + col);
    }

}

void fifo(int* table, int* pf_ind)
{
    int* temp_table = table;
    int* temp_pf_ind = pf_ind;

    int i;
```

```
    for (i = 0; i < RS_len; i++)
    {
        if(exist(temp_table, i, RS[i]) == 0)
        {
            *(pf_ind + i) = 1;
            push(temp_table, i, RS[i]);
        }
        copy_col(temp_table, i);
    }
}

/* starting at col, calculate how many steps to meet "value" again in the future
   Used for optimal */
int distance(int value, int col)
{
    if(col >= RS_len)
    {
        return RS_len + 1; //return a big number
    }

    int distance = 0;
    int i = col;

    while(RS[i] != value)
    {
        distance ++;
        i ++;
        if(i == RS_len)
        {
            return RS_len + 1; // can not find value, return a big number
        }
    }
    return distance;
}

/* return the index to be replaced   0 <= index < M
   Used for optimal*/
int optimal_helper(int* table, int col)
{
    int steps[M];
    int temp_number;
    int i;

    for(i = 0; i < M; i++)
    {
        temp_number = *(table + i * RS_len + col);
        steps[i] = distance(temp_number, col + 1);
    }

    int max_index = 0;
    for(i = 0; i < M; i++)
    {
        if(steps[i] > steps[max_index])
        {
            max_index = i;
        }
    }

    return max_index;
}

void optimal(int* table, int* pf_ind)
{
    int* temp_table = table;
    int* temp_pf_ind = pf_ind;

    int i;
    int replace_index;

    for (i = 0; i < RS_len; i++)
    {
        if(i < M)
```

```
        {
            *(table + i * RS_len + i) = RS[i];
            *(pf_ind + i) = 1;
            copy_col(temp_table, i);
            continue;
        }
        if(exist(temp_table, i, RS[i]) == 0)
        {
            *(pf_ind + i) = 1;
            replace_index = optimal_helper(temp_table, i);
            *(table + replace_index * RS_len + i) = RS[i];
        }
        copy_col(temp_table, i);
    }
}

/* return how many times that "value" is not used starting at "col"
   Used for LRU*/
int unused(int value, int col)
{
    int i = col;
    int distance = 0;
    while(RS[i] != value)
    {
        distance++;
        i--;
        if(i < 0)
        {
            return col + 1;
        }
    }
    return distance;
}

/* return the index to be replaced
   Used for LRU*/
int lru_helper(int* table, int col)
{
    int i;
    int temp_number;
    int steps[M];

    for(i = 0; i < M; i++)
    {
        temp_number = *(table + i * RS_len + col);
        steps[i] = unused(temp_number, col - 1);
    }

    int max_index = 0;
    for(i = 0; i < M; i++)
    {
        if(steps[i] > steps[max_index])
        {
            max_index = i;
        }
    }
    return max_index;
}

void lru(int* table, int* pf_ind)
{
    int* temp_table = table;
    int* temp_pf_ind = pf_ind;

    int i;
    int replace_index;

    for (i = 0; i < RS_len; i++)
    {
        if(i < M)
        {
            *(table + i * RS_len + i) = RS[i];
```

```
            *(pf_ind + i) = 1;
            copy_col(temp_table, i);
            continue;
        }
        if(exist(temp_table, i, RS[i]) == 0)
        {
            *(pf_ind + i) = 1;
            replace_index = lru_helper(temp_table, i);
            *(table + replace_index * RS_len + i) = RS[i];
        }
        copy_col(temp_table, i);
    }
}


int main()
{
    read_file(); //after read_file, all four global variables are prepared
    get_max_digit();
    display_general_info();

    int i;
    int j;
    /* page fault indicator:  1 --> page fault ; 0 --> no page fault */
    int pf_ind[RS_len];
    /* algorithm table */
    int table[M][RS_len];


    /*=============================================FIFO=============================================*/
    for(i = 0; i < RS_len; i++)
    {
        pf_ind[i] = 0;  // assume no page fault at the beginning
    }
    for (i = 0; i < M; i++)
    {
        for(j = 0; j < RS_len; j++)
        {
            table[i][j] = -1;   //initialize to -1 first
        }
    }
    printf("FIFO\n");
    fifo(&table[0][0], &pf_ind[0]);
    display_table(&table[0][0], &pf_ind[0]);
    printf("\n");


    /*============================================Optimal============================================*/
    for(i = 0; i < RS_len; i++)
    {
        pf_ind[i] = 0;  // assume no page fault at the beginning
    }
    for (i = 0; i < M; i++)
    {
        for(j = 0; j < RS_len; j++)
        {
            table[i][j] = -1;   //initialize to -1 first
        }
    }
    printf("Optimal\n");
    optimal(&table[0][0], &pf_ind[0]);
    display_table(&table[0][0], &pf_ind[0]);
    printf("\n");


    /*=============================================LRU=============================================*/
    for(i = 0; i < RS_len; i++)
    {
        pf_ind[i] = 0;  // assume no page fault at the beginning
    }
    for (i = 0; i < M; i++)
    {
```

```
        for(j = 0; j < RS_len; j++)
        {
            table[i][j] = -1;    //initialize to -1 first
        }
    }
    printf("LRU\n");
    lru(&table[0][0], &pf_ind[0]);
    display_table(&table[0][0], &pf_ind[0]);

    return 1;
}
```